99 Swift interview questions to hire top developers

Questions

1. What's the big idea behind Swift being a 'type-safe' language? Can you give a simple example?

2. Imagine you're explaining optionals to someone who's never coded before. How would you do it?

3. Can you explain the difference between a struct and a class in Swift, like I'm five? What are the real-world implications of these differences?

4. What's the deal with 'mutating' functions in structs? Why do we need them?

5. Tell me about 'enums' in Swift. When would you use them instead of just using strings or numbers?

6. What are protocols in Swift? How can they help you write better code?

7. What is protocol-oriented programming? Why is it important in Swift?

8. Explain the difference between 'let' and 'var'. Why would you choose one over the other?

9. What's the point of using 'guard let'? How does it make your code safer?

10. What are closures in Swift? Can you give a simple example of when you might use one?

11. How do you handle errors in Swift? What's the 'do-try-catch' thing all about?

12. What are generics in Swift? Why should we care about them?

13. What is the difference between 'nil' and an empty string in Swift?

14. Can you explain how memory management works in Swift? What is ARC?

15. What are extensions in Swift? How can they be helpful?

16. What are the different access control levels in Swift (e.g., private, public, internal)? Why are they important?

17. Explain the concept of a computed property in Swift. How is it different from a stored property?

18. What are the benefits of using Swift's collection types (arrays, dictionaries, sets)? How do they differ?

19. What's the deal with 'defer' in Swift? When would you use it?

20. How do you perform asynchronous operations in Swift? What are some common techniques?

21. Explain the difference between '==' and '===' in Swift (if there is one, and if not, why?).

22. What are higher-order functions in Swift? Can you name a few common ones?

23. Describe a situation where you might use a delegate in Swift.

24. What are Key-Value Observing (KVO) and Key-Value Coding (KVC) in Swift? When might you use them?

25. How would you typically structure a Swift project? What are some common architectural patterns?

26. What is the purpose of the 'Codable' protocol in Swift?

27. What is a framework? Have you used any frameworks and how was the experience?

28. Can you explain the concept of immutability in Swift and why it's important?

29. What is method swizzling and why should you avoid it?

30. What is dynamic dispatch and static dispatch? How do they relate to Swift?

31. What is the difference between let and var? Can you explain with an example?

32. Imagine you're explaining what an optional is to a friend. How would you describe it and why it's useful?

33. What are the basic data types in Swift (like Int, String, Bool)? Can you give an example of when you might use each?

34. What is a function in Swift? Can you write a simple function that adds two numbers together?

35. What is an Array, and how do you add or remove elements from it?

36. What is a Dictionary, and how is it different from an Array?

37. Explain the concept of a loop in Swift. Give an example of using a 'for' loop.

38. What is the purpose of an 'if' statement? Can you give an example of using one?

39. What is a class in Swift? How does it relate to creating objects?

40. What is the difference between a struct and a class in Swift? When might you choose one over the other?

41. What is inheritance in Swift, and why is it useful?

42. Can you describe what a protocol is and how it is used in Swift?

43. What is the point of using guard statements in Swift?

44. Explain the difference between '==' and '===' in Swift, if any.

45. How do you handle errors in Swift using 'try', 'catch'?

46. What are closures in Swift? Have you used them before?

47. What is a Swift Package? Have you used any?

48. How do you use storyboards to build a user interface in iOS?

49. What is Auto Layout and how does it help create user interfaces that work on different screen sizes?

50. Explain the Model-View-Controller (MVC) design pattern.

51. What is the purpose of using enums in Swift? Can you show me an example?

52. Have you used any debugging tools in Xcode? Which ones do you find the most useful?

53. What are higher-order functions in Swift, and can you provide an example of how you've used one?

54. Explain the difference between `class` and `static` keywords when defining type members.

55. Describe the use cases for `defer` in Swift, and how it affects the control flow.

56. What are Swift's KeyPaths, and how can they be used with KVO?

57. How do you handle errors in Swift, and what are the advantages of using Result type?

58. Explain the concept of property observers (`willSet`, `didSet`) and their use cases.

59. Describe the differences between `Any` and `AnyObject` in Swift.

60. What are generics in Swift, and why are they useful? Provide an example.

61. Explain the use of `inout` parameters in Swift functions.

62. How does Swift's memory management work, and what is ARC?

63. What are protocols in Swift, and how do they enable polymorphism?

64. What are extensions in Swift, and what are their limitations?

65. How do you handle concurrency in Swift using Grand Central Dispatch (GCD)?

66. What are closures in Swift, and how do they capture values?

67. Explain the concept of optionals in Swift and how to unwrap them safely.

68. Describe the use of guard statements in Swift.

69. What are the benefits of using Swift's `Codable` protocol for serialization and deserialization?

70. How do you create and use custom operators in Swift?

71. What are Swift's access control levels (e.g., `private`, `fileprivate`, `internal`, `public`, `open`) and when would you use each?

72. Explain the difference between a `struct` and a `class` in Swift, and discuss their respective use cases.

73. Explain the concept of 'copy-on-write' optimization in Swift and how it affects the performance of value types, using an example with arrays.

74. How does Swift's memory management handle retain cycles in closures, and what techniques can be used to prevent them, such as weak and unowned references?

75. Describe the differences between `inout` parameters and returning a new value from a function. When would you choose one over the other, especially considering performance?

76. What are the advantages and disadvantages of using Swift's 'Codable' protocol for serialization and deserialization, and how does it compare to manual parsing?

77. Explain the use cases for Swift's property wrappers and provide an example of how you might create a custom property wrapper to enforce data validation.

78. Describe the role of Swift's 'Result' type and how it simplifies error handling compared to traditional try-catch blocks, especially in asynchronous operations.

79. How can you leverage Swift's generics to write more reusable and type-safe code, and what are the limitations of generics in certain scenarios?

80. Explain the concept of Protocol-Oriented Programming (POP) in Swift and how it promotes code reusability and testability compared to Object-Oriented Programming (OOP).

81. Describe the differences between 'Any' and 'AnyObject' in Swift and provide examples of when you might use each, considering type safety.

82. How does Swift handle concurrency using Grand Central Dispatch (GCD), and what are some common pitfalls to avoid when working with concurrent code?

83. Explain the use of Key-Value Observing (KVO) in Swift and how it can be used to observe changes in object properties, considering its performance implications.

84. Describe the purpose of Swift's 'defer' statement and how it ensures that code is executed regardless of how a function exits, including error handling.

85. How does Swift's optional chaining work, and what are some scenarios where it can simplify code while avoiding potential runtime crashes?

86. Explain the concept of associated types in protocols and how they allow you to define protocols that work with generic types, ensuring type safety.

87. Describe the use cases for Swift's 'dynamic' keyword and how it affects the behavior of method dispatch at runtime, considering its impact on performance.

88. How can you use Swift's Combine framework to handle asynchronous events and data streams, and what are the advantages of using Combine over traditional delegation patterns?

89. Explain the concept of method swizzling in Swift and how it can be used to modify the behavior of existing methods at runtime, considering its potential risks.

90. Describe the role of Swift's 'autoreleasepool' and how it manages memory for Objective-C objects in Swift code, especially when interoperating with legacy code.

91. How does Swift handle bridging between Swift types and Objective-C types, and what are some common challenges to consider when working with mixed codebases?

92. Explain the concept of SwiftUI's 'Environment' and how it allows you to pass data and dependencies down the view hierarchy, considering its impact on testability.

93. Describe the differences between 'ObservableObject' and '@StateObject' in SwiftUI, and when you would use each for managing state in your views.

94. How can you use Swift's actors to safely manage concurrent access to shared mutable state, preventing data races and ensuring thread safety?

95. Explain the purpose of Swift's opaque return types (using 'some View' in SwiftUI) and how they allow you to hide implementation details while still ensuring type safety.

96. Describe the use of Swift's new concurrency features like 'async' and 'await' and how they simplify asynchronous programming compared to GCD and closures.

97. How can you leverage Swift's macro system to generate code at compile time, and what are the benefits of using macros for code reduction and performance optimization?

98. Explain the concept of existential types (e.g., `any Protocol`) in Swift and how they differ from generics and concrete types, particularly in the context of protocol conformance.

99. Describe the advantages and disadvantages of using Swift's Core Data framework for data persistence, and how it compares to other options like Realm or SQLite.